İZMİR UNIVERSITY OF ECONOMICS

SE 322 - SE 318
SOFTWARE VERIFICATION AND VALIDATION
SPRING 2023-2024

CUSTOMERFEEDBACK-MS

ILKER EKIN ERDOĞDU

MUHAMMET ERSIN KARAER

BILGEHAN GÖKSÜ ÜNSAL

ÖZGÜR ÇOLAK

MERT KURT

UNIT TEST DOCUMENT

Version *<3.0>*

<31.05.2024>

VERSION HISTORY

VERSION 1.0 (03.04.2024)
In this release version, we implemented our high and moderate requirements for the project.
VERSION 2.0 (02.05.2024)
In this release version, we implemented the missing high and moderate requirements for the project. Furthermore, we performed the tests for the requirements which we implemented. There are total 15 test class implemented for to check if the requirements we implemented working correctly.
VERSION 3.0 (30.05.2024)
In this release version, we implemented the missing low requirements, perform some quality implementation to previous code. Also, we created extra test classes to be able to test the new implemented requirements are working correctly or not.

1 INTRODUCTION

1.1 PURPOSE OF THE TEST CASE DOCUMENT

The Test Case document documents the functional requirements of the *<test case title>* test case. The intended audience is the project manager, project team, and testing team. Some portions of this document may on occasion be shared with the client/user and other stakeholder whose input/approval into the testing process is needed.

Test cases are designed to verify that the application is operating as expected. Test case writers design test cases so testers can determine whether an app or software system's feature is working correctly.

1.2 CONSTRAINTS

Programming Language: Java
Unit Test Framework: JUnit 5

2 UNIT TEST FRAMEWORK: JUNIT

Junit is a widely used testing framework for Java applications. Junit has annotations such as "@Test,@Before,@After". Also, Junit has assertions like "assertEquals,assertTrue,assertFalse,assertNull and assertNotNull". Junit providing a flexibility in organizing tests with "Test Suites". Conclusion, we can say that Junit is simplifies the process of writing and executing tests for developers/testers.

3 TEST CASES

Test Case 1	
Test Definition	
Test if getAllCategoryNames functionality is working	
Input Value	
“Example Category”,”Example Category 2”,”Example Category 3”	
Expected Value	**Actual Value**
"Example Category", "Example Category 2", "Example Category 3"	**"Example Category", "Example Category 2", "Example Category 3"**
Result of Test Case	***succesfull***
Test Script	

```
public void testGetAllCategoryNames() {
    // Retrieve the actual array of category names which are added statically
    String[] actual = getAllCategoryNames();
System.out.println(Arrays.toString(actual));
    // Define the expected array of category names
    String[] expected = {"Example Category", "Example Category 2", "Example Category 3", "Test Category", "Now I am dont"};
    // Perform the assertion
    assertArrayEquals(expected, actual, "The categories should match the expected values.");
  }
```

Test Case 2	
Test Definition	
Test if user can add non-existing category	
Input Value	
“Test Category”	
Expected Value	**Actual Value**
Succesfuly category added	**Succesfully category added**
Result of Test Case	***successful***
Test Script	

```
public void addNonExistingCategory()
{
      assertTrue(addCategory("Test
Category"));
   }
```

Test Case 3	
Test Definition	
Test if user can add existing category	
Input Value	
“Example Category”	
Expected Value	**Actual Value**
“A category Exists with the same name”	**“A category Exists with the same name”**
Result of Test Case	***successful***
Test Script	

```
Public void addExistingCategory()
{
AssertFalse(addCategory(“Example
Category”));
}
```

Test Case 4	
Test Definition	
Test if user can remove non-existing category	
Input Value	
“Non existed Category”	
Expected Value	**Actual Value**
Category does not exist	**Category does not exist**
Result of Test Case	***successful***
Test Script	

```
public void removeNonExistingCategory() {

assertFalse(removeCategory("Non Existed Category"));
   }
```

Test Case 5	
Test Definition	
Test if user can remove existing category	
Input Value	
Example Category 4	
Expected Value	**Actual Value**
“Example Category 4 removed”	**“Example Category 4 removed”**
Result of Test Case	***successful***
Test Script	

```
Public void removeExistingCaregory(){
AddCategory(“Example Category 4”);
AssertTrue(removeCategory(“Example category 4”));
}
```


Test Case 6	
Test Definition	
Test if user edit non existing category	
Input Value	
Non existed category, maybe existed category	
Expected Value	**Actual Value**
Category does not exist	**Category does not exist**
Result of Test Case	***succesful***
Test Script	
public void editNonExistCategory() { **assertFalse(editCategory("Non existed category", "Maybe existed category"));** **}**	
Test Case 7	
Test Definition	
Test if user can edit the exist category	
Input Value	
I am here	
Expected Value	**Actual Value**
Now I am dont	**Now I am dont**
Result of Test Case	***successful***
Test Script	

```
Public void editExistCategory(){
AddCategory("I am here");
AssertTrue(editCategory("I am here", Now I am dont"));
String [] array = getAllCategoryNames();
List<String> arrayList = new ArrayList<>(Arrays.asList(array));
AssertFalse(arrayList.contains("I am here"));
AssertTrue(arrayList.contains("Now I am dont"));
}
```

Test Case 8	
Test Definition	
Test if category exist	
Input Value	
"Example Category"	
Expected Value	Actual Value
Category exist	Category exist
Result of Test Case	*successful*
Test Script	

```
public void isCategoryExistTest() {

assertTrue(isCategoryExist("Example Category"));
   }
```

Test Case 9
Test Definition
Test if category non exist
Input Value
I'm not exist in here

Expected Value	**Actual Value**
Category does not exist	**Category does not exist**
Result of Test Case	***successful***
Test Script	

Public void isCategoryNonExistTest(){
AssertFalse(isCategoryExist("I'm not exist in here"));
}

Test Case 10	
Test Definition	
Test if user can add new user	
Input Value	
"Test User","testpass","testname","testsurname"	
Expected Value	**Actual Value**

New user added successfuly	New user added successfuly
Result of Test Case	*successful*
Test Script	

public void addNewUserTest() {
assertTrue(addNewUser("Test User", "testpass", "testname", "testsurname"));
}	

Test Case 11	
Test Definition	
Test if user can add existing user	
Input Value	
“ekaraer”,”password”,”Ersin”,”Karaer”	
Expected Value	Actual Value
An user exists with the same username	An user exists with the same username
Result of Test Case	*successful*
Test Script	

Public void addExistingUserTest(){
AssertFalse(addNewUser(“ekaraer”,”password”,”ersin”,”karaer”));
}	

Test Case 12	
Test Definition	
Test if user can add new user with missing parameters	
Input Value	
Null, “password”,”x”,”y”	
Expected Value	**Actual Value**
Missing parameters	**Missing parameters**
Result of Test Case	***succesful***
Test Script	

```
public void addNewUserMissingParameters() {
        assertThrows(RuntimeException.class,        ()        ->
addNewUser(null, "password", "x", "y"));
        assertThrows(RuntimeException.class,        ()        ->
addNewUser("newuser", null, "x", "y"));
        assertThrows(RuntimeException.class,        ()        ->
addNewUser("newuser", "password", null, "y"));
        assertThrows(RuntimeException.class,        ()        ->
addNewUser("newuser", "password", "x", null));
    }
```

Test Case 13	
Test Definition	
Test the password authorization for existing user	
Input Value	
“ekaraer”	
Expected Value	**Actual Value**
pass	**pass**
Result of Test Case	***successful***
Test Script	

```
Public void gerPasswordAndAuthorizationExistingUserTest(){
String[] result = getPasswordAndAuthorization(“ekaraer”);
AssertNotNull(result);
AssertEquals(“pass”,result[0]));
AssertEquals(Constants.AUTHENTICATED_ADMIN,result[1]);
}
```

Test Case 14	
Test Definition	
Test the password authorization for non existing user	
Input Value	
Non existing user	
Expected Value	**Actual Value**
User does not exist	**User does not exist**
Result of Test Case	***successful***
Test Script	

```
public                                                    void
getPasswordAndAuthorizationNonExistingUserTest()
{
      assertNull(getPasswordAndAuthorization("Non
existed user"));
   }
```

Test Case 15	
Test Definition	
Test getPasswordAndAuthorizationMissingParameters function is works	
Input Value	
null	
Expected Value	**Actual Value**
null	**null**
Result of Test Case	***successful***
Test Script	

```
Public                                                    void
getPasswordAndAuthorizationMissingParameters(){
AssertNull(getPasswordAndAuthorization(null));
}
```

Test Case 16
Test Definition
Test if user can share feedback positive
Input Value
“Feedback 4”,”Feedback 4 content”,”Example Category”,”2”

Expected Value	Actual Value
4	1
Result of Test Case	***fail***
Test Script	

```
public void testShareFeedbackPositive() {
        Actions.signIn("user", "pass");

        boolean shared = Actions.shareFeedback("Feedback 4", "Feedback 4 content", "Example Category", 2);
        assertTrue(shared);

        String[] titles = Actions.getAllFeedbackTitlesOfAuthenticatedUser();
        assertNotNull(titles);
        // there are 3 feedbacks added statically
        assertEquals(4, titles.length);
        assertEquals("Feedback 4", titles[3]);
    }
```

Test Case 17
Test Definition
Test if user can share feedback negative
Input Value
“user1”,”Feedback 1”,”Duplicate content”,”6”

Expected Value	Actual Value
4	4
Result of Test Case	***successful***
Test Script	

```
Public void testShareFeedbackNegative(){
```

```
Actions.shareFeedback(“user1”,”Feedback1”,”Duplicate
content”,6);
}
```

Test Case 18	
Test Definition	
Test if user can sign out positive	
Input Value	
-	
Expected Value	**Actual Value**
User signed out	**User signed out**
Result of Test Case	***successful***
Test Script	
public void testSignOutPositive() { **assertTrue(Actions.signOut(Actions.getAuthenticatedUsername()));** **}**	
Test Case 19	
Test Definition	
Test if user can sign out negative	
Input Value	
-	
Expected Value	**Actual Value**
null	**null**
Result of Test Case	***successful***
Test Script	
Public void testSignOutNegative(){ **Actions.signOut(null);**	

```
}
```

Test Case 20	
Test Definition	
Test if user can get all feedback titles positive	
Input Value	
“Feedback1”,”Feedback2”,”Feedback3”	
Expected Value	**Actual Value**
“Feedback1”,”Feedback2”,”Feedback3”	**“Feedback1”,”Feedback2”,”Feedback3”**
Result of Test Case	***successful***
Test Script	

```
public void testGetAllFeedbackTitlesPositive() {
      String[]                titles                =
FeedbackRepository.getAllFeedbackTitles();
      assertEquals(3, titles.length);

assertTrue(Arrays.asList(titles).contains("Feedback
1"));

assertTrue(Arrays.asList(titles).contains("Feedback
2"));

assertTrue(Arrays.asList(titles).contains("Feedback
3"));
   }
```

Test Case 21	
Test Definition	
Test if user can get all feedback titles negative	
Input Value	
“Feedback1”,”Feedback2”	
Expected Value	**Actual Value**
“Feedback1”,”Feedback2”	**“Feedback1”,”Feedback2”**
Result of Test Case	***successful***
Test Script	

```
Public void testGetAllFeedbackTitlesNegative(){
FeedbackRepository.feedbacks.clear();
String[] titles =
FeedbackRepository.getAllFeedbackTitles();
AssertEquals(0,titles.length);
}
```

Test Case 22	
Test Definition	
Test if user can get all feedback titles of authenticated user positive	
Input Value	
“Feedback1”,”Feedback2”	
Expected Value	**Actual Value**
“Feedback1”,”Feedback2”	**“Feedba could find”**
Result of Test Case	***fail***
Test Script	

```
public void testGetAllFeedbackTitlesOfAuthenticatedUserPositive() {
      String[] user1Titles =
FeedbackRepository.getAllFeedbackTitlesOfAuthenticatedUser("user1");
      assertNotNull(user1Titles);
      assertEquals(2, user1Titles.length);
      assertTrue(Arrays.asList(user1Titles).contains("Feedback 1"));
      assertTrue(Arrays.asList(user1Titles).contains("Feedback 2"));
   }
```

Test Case 23	
Test Definition	
Test if user can get all feedback titles of authenticated user negative	
Input Value	
nonexistentUser	
Expected Value	**Actual Value**

Feedback could not find	Feedba could find
Result of Test Case	*success*
Test Script	

Public void testGetAllFeedbackTitlesOfAuthenticatedUserNegative(){
String[] nonexistentUserTitles = FeedbackRepository.getAllFeedbackTitlesOfAuthenticatedUser("nonexistentUser");
AssertNull(nonexistentUserTitles);
}	

Test Case 24	
Test Definition	
Test if user can get all feedback details negative	
Input Value	
"nonexistent feedback"	
Expected Value	Actual Value
Feedback could not find	Feedback could not find
Result of Test Case	*successful*
Test Script	

Public void testGetFeedbackDetailsNegative(){
String details = FeedbackRepository.getFeedbackDetails("nonexistent Feedback");
}	

Test Case 25	
Test Definition	
Test if user can get status of the feedback positive	
Input Value	
"user1","Feedback1","Resolved"	
Expected Value	Actual Value
"user1","Feedback1","Resolved"	Feedback could not find
Result of Test Case	*fail*

Test Script
public void testSetStatusOfFeedbackPositive() { **boolean statusSet = FeedbackRepository.setStatusOfFeedback("user1", "Feedback 1", "Resolved");** **assertTrue(statusSet);** **Feedback feedback = getFeedbackByTitle("Feedback 1");** **assertNotNull(feedback);** **assertEquals("Resolved", feedback.getStatus());** **}**

Test Case 26	
Test Definition	
Test if user can set status of feedback negative	
Input Value	
“user1”,nonexistent feedback”,”Resolved”	
Expected Value	**Actual Value**
responded	**responded**
Result of Test Case	***successful***
Test Script	
public void testSetStatusOfFeedbackNegative() { **boolean statusSet = FeedbackRepository.setStatusOfFeedback("user1", "Nonexistent Feedback", "Resolved");** **assertFalse(statusSet);** **}**	
Test Case 27	
Test Definition	
Test if user can response feedback positive	
Input Value	
“Feedback1”,”Response to feedback1”	
Expected Value	**Actual Value**
responded	**responded**

Result of Test Case	***successful***
Test Script	
Public void testResponseFeedbackPositive(){ **Boolean responded = FeedbackRepository.responseFeedback(“Feedback 1”,”Response to Feedback1”);** **Feedback feedback = getFeedbackByTtitle(“Feedback 1”);** **assertNotNull(feedback);** **assertTrue(feedback.getResponses().contains(“Response to feedback 1”));** **}**	

Test Case 28	
Test Definition	
Test if user can response feedback negative	
Input Value	
“Nonexistent Feedback”,”Response to feedback”	
Expected Value	**Actual Value**
Feedback could not find	**Feedback could not find**
Result of Test Case	***successful***
Test Script	
public void testResponseFeedbackNegative() { **boolean responded = FeedbackRepository.responseFeedback("Nonexistent Feedback", "Response to Feedback");** **assertFalse(responded);** **}**	
Test Case 29	
Test Definition	
Test if user can set status of feedback positive with exception	
Input Value	
“F99”,”Some thoughts”,”Example category”,5	

Expected Value	Actual Value
“F99”	“F99”
Result of Test Case	***successful***
Test Script	

Public void testSetStatusOfFeedbackPositive(){
Actions.signIn(“ekaraer”,”pass”);
Actins.shareFeedback(“F99”,”Some thoughts”,”Example category”,5);
AssertTrue(Actions.setStatusOfFeedback(Actions.getAuthenticatedUsername(), “F99”, Constants.FEEDBACK_STATUS_FIXED));
}	

Test Case 30	
Test Definition	
Test if user can set status of feedback negative with exception	
Input Value	
“F99”,”Some thoughts”,”Example category”,5	
Expected Value	**Actual Value**
“F99”	“Some invalid status”
Result of Test Case	***successful***
Test Script	

```
@Test(expected = RuntimeException.class)
  public void testSetStatusOfFeedbackNegative() {
     // sign in as admin
     Actions.signIn("ekaraer", "pass");
     Actions.shareFeedback("F99", "Some thougts", "Example
Category", 5);

Actions.setStatusOfFeedback(Actions.getAuthenticatedUsername(),
"F99", "SOME INVALID STATUS");
  }
```

4. CONCLUSION

To be conclude, we learned how to use perform test for a software. We learned Junit framework, test suite and lot more test application with the help of this

project.
In this project we implemented low,moderate and high requirements for customerFeedback-MS and also we performed the unit test for each requirement ve implemented.